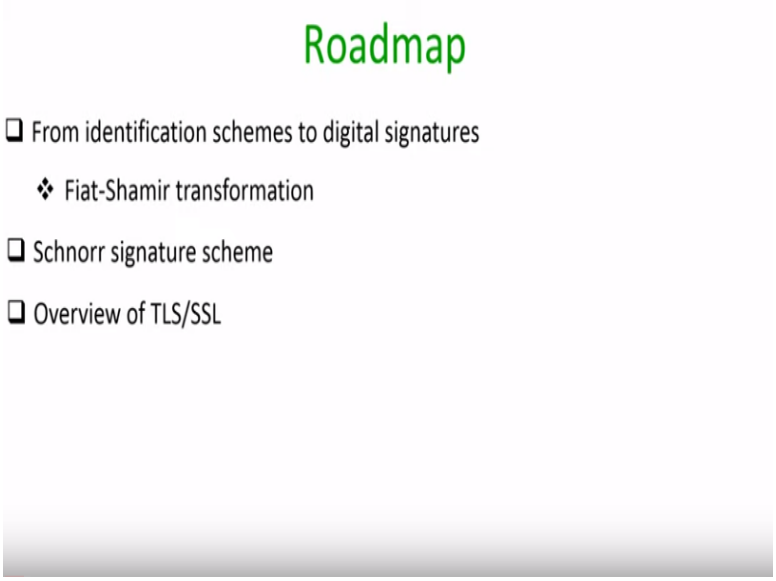
Foundations of Cryptography
Prof. Dr. Ashish Choudhury
Department of Computer Science
Indian Institute of Science– Bangalore

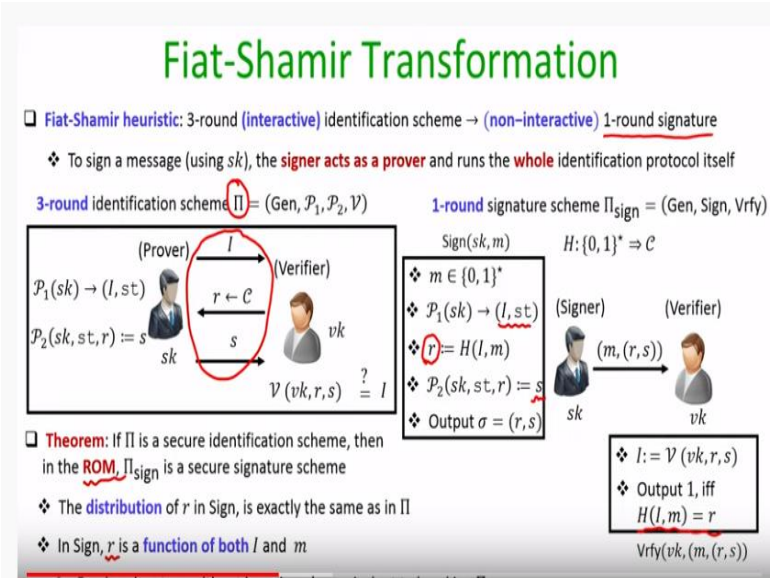
Lecture – 54
Schnorr Signature Scheme and TLS or SSL

(Refer Slide Time: 00:33)



Hello everyone. Welcome to this lecture. The plan for this lecture is as follows and this lecture we will discuss a very powerful transformation or heuristic, which we call as Fiat-Shamir heuristic, which allows us to obtain digital signature schemes from any secure identification schemes and then we will see how to apply this transformation to no signature scheme to obtain a digital signature scheme based on the discrete log assumption and finally we will see an overview of TLS and SSL protocol.

(Refer Slide Time: 01:00)



So, let us start our discussion with Fiat-Shamir transformation. So, what Fiat-Shamir heuristic or transformation does is, it takes any 3-round interactive identification scheme, which has structure of commitment, challenge, response, and the transformation converts that interactive protocol into a non-interactive protocol, non-interactive in the sense that it will have only one round of interaction from the prover to the verifier and using that non-interactive protocol.

Basically we end up getting a 1-round signature scheme and the idea behind this transformation is that. If you have an identification scheme and if you want to get a signature scheme out of that, then to sign the message using the signing key, the signer has to act as a prover and it has to run the whole protocol of the identification scheme in its mind and as per the rules of this transformation, and come with the whole transcript in a single shot and give it to the verifier.

That means, in the 3-round interactive protocol which was there for the identification scheme, the challenge would have been picked by the verifier, but what we are saying here is that after doing this transformation, even the challenge also will be picked by the prover itself. It has to play the role of the prover as well as the verifier simultaneously. So, you might be wondering that if prover is given to pick the challenge on behalf the verifier, the malicious prover or the cheating prover can cheat.

It may not pick a uniformly random challenge. It turns out that the Fiat-Shamir transformation is so clever that it ends up even forcing or cheating prover to pick up

uniformly random challenges. So, here is how we do the conversion of 3-round interactive protocol into 1-round of interactive protocol. So, we are given identification scheme, which has its key generation algorithm, 2 algorithms for the prover and one algorithm for the verifier and using it we want to obtain 1-round signature scheme, which should have its own generation key algorithm, signing algorithm, and a verification algorithm.

As part of this transformation, we assume that we have the public description of some hash function, which maps arbitrary long bit strings to the elements of the challenge space of the underlying identification scheme. So, the key generation algorithm of this signature scheme, will be the same as the key generation algorithm of your identification scheme. Now the signing algorithm is as follows.

Imagine, there is a signer who has the signing key sk, and it has a message on which it wants to generate its signature. So, as I said earlier it has to play the role of the prover as well as the verifier of the identification scheme in itself, so it runs the P1 algorithm on the signing key to obtain the commitment I and a state information, and now it has to pick the challenger randomness r on behalf of the verifier itself.

So, ideally it should pick the challenger randomness r uniformly randomly from the challenge space, but now we are going to design a signature scheme, so the message also should come into the picture while picking the randomness r or the challenger r. So, to pick the challenge r what the signer does here is it runs the hash function H, on the commitment I and the message, which is going to sign. So the idea here is basically, the challenge r is going to be somehow associated with the message, which the sender wants to sign here.

And if we assume or if we model the underlying hash function as a random oracle here, it turns out that the value r, which signer is going to obtain by generating r as per this method, will be indeed a uniformly random value from the challenge space of the underlying identification scheme. Once the r is generated, the signer has to generate the response part, namely by running the algorithm P sub 2, and finally the signature is r, s.

So, now to send a signed message to the verifier, it will just send a message along with the signature r, s and the way verifier verifies the signature is, it runs the verification algorithm of the underlying identification scheme, but there is a difference. In the original identification

scheme, the verifier was verifying whether the output of the verification algorithm with the verification key and the r and s part of the transcript gives the I part of the transcript or not.

Here on the other hand, the s part of the verification algorithm of the signature scheme, the verifier is going to run the verification algorithm of the underlying identification scheme, but it does not compare the output of the V function with the I component, rather because it does not know the I component as it is. If you see the structure of this signature scheme, the I is not given by the signer to the verifier.

Whereas in the identification scheme, the verifier would have already obtained a I and that is why it could compare the output of the v function with the commitment I given by the prover. So, in the verification algorithm of the signature scheme, I is computed from scratch by the verifier by running the V function and ideally this I should be the same I, which legitimate prover in the identification scheme would have committed to an honest verifier.

Now, to verify the signature what the verifier does is, it has now the I value, it has now the m value. It queries the random oracle, and check whether the output of the random oracle on I and m, gives the r part of the signature or not and if it indeed gives the value of r, then it accepts the signature, otherwise, it rejects the signature. Ideally, if indeed the signer and the signer is honest, then the I generated by running the verification function or the v function on verification key r and s would have given an I, such that the output of the random oracle on the generated I and m should indeed be equal to r.

So, the theorem statement that we can prove here is that if you are in the random-oracle model, and if this identification scheme pi is a secure identification scheme, then the signature scheme that we have generated indeed here is a secured signature scheme, and the prove slightly involved is that we go into the technicality of the random-oracle model, so I am skipping the accompanied formal proof, we will just see an overview of the proof, you can see the complete formal proof and the (()) (08:09) cylinder.

The idea here is that if you see the distribution of r that the signer has generated in the signature algorithm, it is distribution is exactly the same as it would have been in a real execution of the identification scheme, provided you are in the random-oracle model. That

means, even the signer is corrupt, it has no control over the randomness r because it is going to be a uniformly random value, because it is going to be an output of the random oracle.

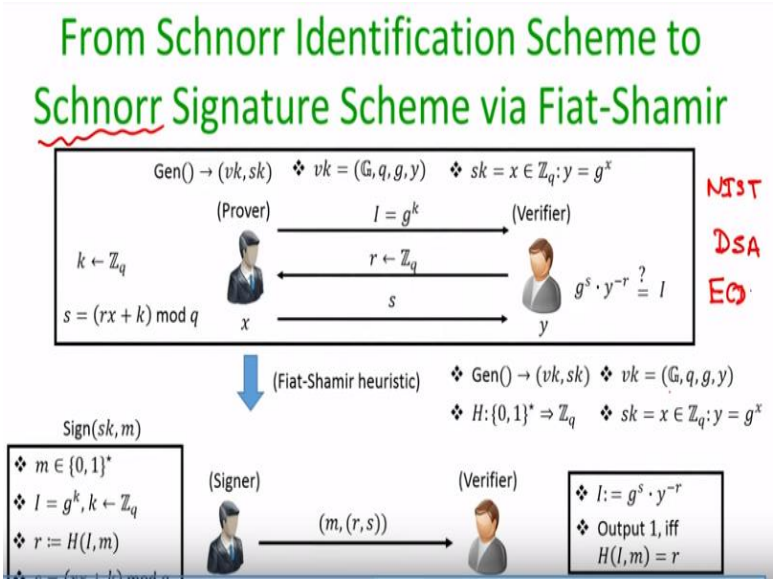
The way we have designed the signature scheme, in the signing algorithm, the randomness r, or the challenge r, which the signer is generating, is basically a function of both the commitment I as well as the message on which it wants to generate the signature. So, if we have an adversary who wants to forge a signature on behalf of signer without actually knowing the signature key sk, then basically what the forger has to do is without knowing the signing key sk.

It has to come up with some I and state information and it has to come up with some challenge r, by querying the random oracle, and it has to come up with some response s, such that overall it is an accepting transcript, namely I, r, and s is an accepting transcript and not only that output of the random-oracle on the commitment I and the message is equal to r. This whole thing has to do without actually knowing the signing key sk.

If there exist an adversary who can indeed do that with a significant probability, then intuitively it mean that it have an adversary who can even forge or who can break the security of this underlying identification scheme, namely, even without knowing the signing key or the secret key, sk, that adversary could come up with a legitimate accepting transcript on the behalf of the prover and convince the verifier and get it accepted.

But that goes against the assumption that underlying identification scheme is a secure identification scheme. So that is a intuitive idea behind this Fiat-Shamir Transformation. I am not going into the full formal details.

(Refer Slide Time: 10:20)



Now, let us see how we can apply the Fiat-Shamir transformation on the identification scheme due to Schnorr and as a result, we obtain a signature scheme, which we call as Schnorr Signature Scheme. I stress that this Fiat-Shamir transformation is a generic transformation. It can be applied on any identification scheme, on any three-round identification scheme, to obtain a 1-round signature scheme. Here we are applying it in the context of the Schnorr Identification Scheme.

So, recall that in the Schnorr Identification Scheme, the secret key was the discrete log x of a publicly known value y, and the identification scheme was as follows. The prover in the identification scheme wants to know the discrete log x, so to prove that it commits a randomly chosen value k, by submitting g to the power k, the verifier throws random linear combined little r and the prover has to come up with a response s.

Namely a linear combination of the discrete log x and the value k, which it has committed and the verifier verifies the g to the power x multiplied by y to power –r is equal to the commitment of the prover or not. If you apply the Fiat-Shamir heuristic on the identification scheme, the key generation algorithm of the signature scheme will be as follows. The verification key will be the description of the cyclic group, the size of the cyclic group, the generator and the uniformly random group element y where the corresponding discrete log x is the signing key.

Along with that there will be a publicly known description of a hash function mapping bit strings to the elements of Zq, because the challenge space here is nothing but Zq and the

signing algorithm will be as follows. So, imagine there is a signer with a discrete log x to sign a message m, it first computes the commitment by picking a randomly chosen value k, and then it picks the challenge on behalf of the verifier by calling the random-oracle H on the inputs I and message m, and then with respect to the challenge r, it computes the response namely the linear combination of x and k, and the signature is r, s and the message m.

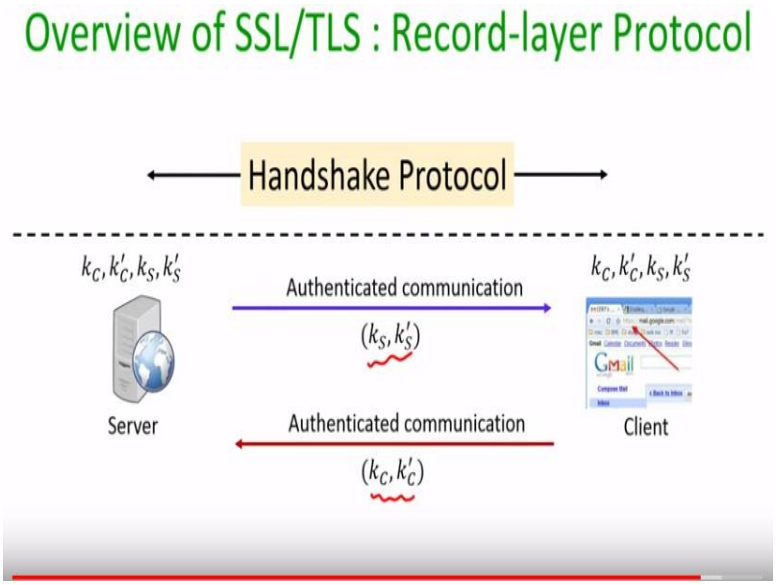
To verify the signature, the verifier has to generate the commitment I, so to generate the commitment I, it computes g to the power s, y to the power –r, and whatever comes out, that is treated as the commitment of the signer for the underlying Schnorr identification scheme and then the verifier generates the challenge which this computed commitment and the message which the signer is sending would have produced.

So for that the verifier calls random-oracle on the computed commitment on the message and checks whether the output of the random oracle gives the challenge r, which the signer is claiming to be a part of the signature. As per the security of the Fiat-Shamir heuristic, this 1-round signature scheme is indeed a security signature scheme in the random-oracle model. So, now we have a candidate signature scheme based on the discrete log assumption.

It turned out that Schnorr patented this idea of. So, he basically patented his three round identification scheme, so as a result the signature scheme that comes out by applying the Fiat-Shamir transformation is also considered as patented, so we cannot use it in the public domain. So what NIST did is, it developed a variation of Schnorr Identification scheme and by applying the Fiat-Shamir transformation on that variant, it obtained a signature scheme based on the discrete log assumption, which we call as DSA, Digital Signature Algorithm.

It is a well-known standard which we use in practice and the underlying group which we use in this DSA algorithm is the group based on the point of elliptic curves modulo prime, then the instantiation of the DSA is called as ECDSA, Elliptic Curve Digital Signature Algorithm and this is a very popularly used digital signature algorithm used in several real world applications.

(Refer Slide Time: 14:48)



So, now more or less we come to the end of the discussion on public key cryptography. We have seen instantiation of encryption scheme, digital signatures and so on and during the first half of the course, we have regressively discussed about symmetric key encryption, so now we will see that how combining various cryptographic primitives in both the worlds. We come up with real world secure communication protocol, so what I will discuss is a very high level overview of the SSL protocol, which is actually the successor of the TLS protocol and this is just a very high level overview.

We should not treat the discussion as a real implementation of the SSL for the exact details under complete analysis of the SSL protocol. You should refer to any standard test in the network security. So, here is how the SSL protocol works. So, imagine you have a computer and in your browser, you type https followed by some xyz.com, say google.com. So as soon as you type https, the s here denotes that you actually want to initiate a secure communication session with the server called mail.google.com and as soon as you type this actually the SSL protocol starts executing, and as part of this SSL protocol.

There are two sub protocols, which get executed, there is a handshake protocol where some interaction happens between the client, which the browser in this case and server which is google.com in this case and as far as this handshake protocol as far as the handshake protocol if everything goes fine, then a key is established between the server and the client. So, to begin with the client, and the server had no pre-shared information but as soon as the handshake protocol gets executed and authenticated key exchange happens between the server and the client.

If the handshake protocol is successful, the key will be successfully established between the server and the client. Now, once the key is established, the second sub protocol that gets started executing is the record-layer protocol and this record-layer protocol what it does is does the it does the authenticated private communication between the server and the client using the keys which have been established during the handshake protocol.

In terms of cryptography primitives to do the authenticated key exchange, the handshake protocol uses public key cryptography whereas once the key has been established to do the actual communication between the server and the client, the record-layer protocol uses the private key primitives. Now you can see that how exactly the various cryptographic primitives gets combined to do or solve the whole problem of secure communication here.

So now let us go a little bit deeper into the details of the handshake protocol and record-layer protocol. So, remember the goal of handshake protocol is to do an authenticated key exchange between the client and the server and imagine that the server has its own public key and a decryption key for a CCS secure key encapsulation mechanism and imagine that we have various certificate authorities available in the market who have their own signing key and verification key.

In this example, I am assuming that there are four certificate authorities, and I assume here that the verification key of the certificate authorities are already preconfigured in the browser, which is used by the client. So, if you want you can go to the settings of the browser that you are using and you can see the verification certificate of the well-known certificate authorities, which are already embedded in your browsers.

I also assume here that the server in this case has a certificate issued by the second certificate authorities certifying its encryption key or the public key and such a certificate is available with the server. So, as soon as the handshake protocol starts executing between the client and the server, the first message goes from the client to the server, were basically client sends the supported ciphersuites namely what version of the hash function it supports, what version of block ciphers it supports, what version of pseudorandom generator it supports, and so on and along with that the client sends the randomly chosen nonce value.

In response, the server picks a random nonce, and it sends what ciphersuites it supports, namely its version of hash function, its version of block ciphers, pseudorandom generators and so on, and along with that the server sends the public key of its key encapsulation mechanism, and to convince that indeed this public key belong to the so called server, the certificate sends the certificate which it might have obtained from one of its certificate authorities.

In this example, it is the second certificate authority. So, basically the idea here is that both client and server are trying to agree upon the version of ciphersuites that they are going to use for the rest of the communication, and along with that the server is sending its encryption key or the public key and prove that indeed the public key belong to the server by sending the corresponding certificate.

Now, what the client does is, it checks or it verifies the certificate and to verify the certificate, it uses the verification key of the certificate authority who has issued that certificate to the server, and in this case the certificate was issued by the second certificate authority. The verification key of the second certificate authority will be used and only the certificate is successfully verified, the communication will proceed further, otherwise the client will abort the session there itself.

Now, it might be possible that the certificate is issued by a certificate authority whose verification key is not embedded in the browser of the client. In that case, this verification will have no output and as a result, the browser will throw a warning message to the user and to the client namely saying that, I cannot identify or I cannot recognize the validity of the certificate and this is a common error message, which we very of an encounter which we try to do an SSL communication with a server, whose certificate cannot be recognized by our browser.

We get the warning message that you can proceed at your own risk. That is why it is always recommended to update the versions of your browser because every time you come up with your new version of your browser, the verification key of new certificate authorities get embedded in the new versions and hence you will not get this kind of error message. For this example, I assume that the verification key of the certificate authority is available with the client and the certificate is verified properly.

Once the certificate verification happens, now what client does is, it runs the encapsulation algorithm of the underlying key encapsulation mechanism using the public key provided by the server, and as a result it will output a secret key, which I denote as pmk, and the encapsulation of this pmk. This pmk is nothing but premaster key, so this encapsulation algorithm will output a premaster key and its encapsulation c.

The encapsulation c will be sent to the server and what the client is going to do is, it is going to run a key derivation function on the inputs Nc and Ns, namely the two nonces, which are picked by the client and the server respectively and the premaster key and output is considered as a master key. Moreover, whatever communication has happened between the client and the server till now including this encapsulation c, let it be called as transcript.

This whole transcript client authenticates by using a message authentication code and using the master key as the key, and the corresponding tag is sent to the server. So, that is a message now from the client side. What the server does is, it takes the encapsulation c and runs the decapsulation algorithm on this encapsulation c to obtain the same premaster key pmk, and once it obtains the premaster key, it runs the key derivation function on the two nonce values and the premaster key to obtain the master key.

So, now as per our assumption both client and server would have agreed upon the versions of the key derivation function that they are going to use because that is the part of the supported cipher suites. It ensures that both the client and server are using the same version of the key derivation function and the same version of the key encapsulation mechanism. Now, the server additionally does the following step.

So, it has seen now the authentication of the whole transcript that client has sent to it, so it verifies the tag on the transcript that the client is sending to it and it aborts the tag verification phase, whereas the tag verification is successful, it does the following. It runs a p pseudorandom generator on the master key and derive four keys, which are denoted by kc, kc dash, k sub s, k sub s dash.

In addition, whatever transcript now the server has seen till now, we call it as transcript dash and this includes all the thing that client has sent to it followed by the authentication of the

transcript that client has sent to the server. This whole thing I call it as transcript prime and this transcript prime is now authenticated by the server using the master key mk and tag is sent to the client.

What client is going to do is, it will perform a verification on the transcript prime using the master key, if the verification fails, it aborts, otherwise it also runs the same pseudorandom generator, which was executed by the server on the master key mk, and derive the same four keys, which server has generated and this completes the handshake protocol. Now, you can see in this whole handshake protocol, we use the public key primitive namely the key encapsulation mechanism, and along with that we are using the key derivation function and a pseudorandom generator.

Till the end of the handshake protocol, actual encryptions of the messages, which the client would like to communicate to the server has not happened. Till now, only the key exchange has happened and the key exchange only invokes the key encapsulation mechanism along with the key derivation function and the pseudorandom generator. Now, assuming that the handshake protocol is successfully executed, both server and client have now two pairs of keys.

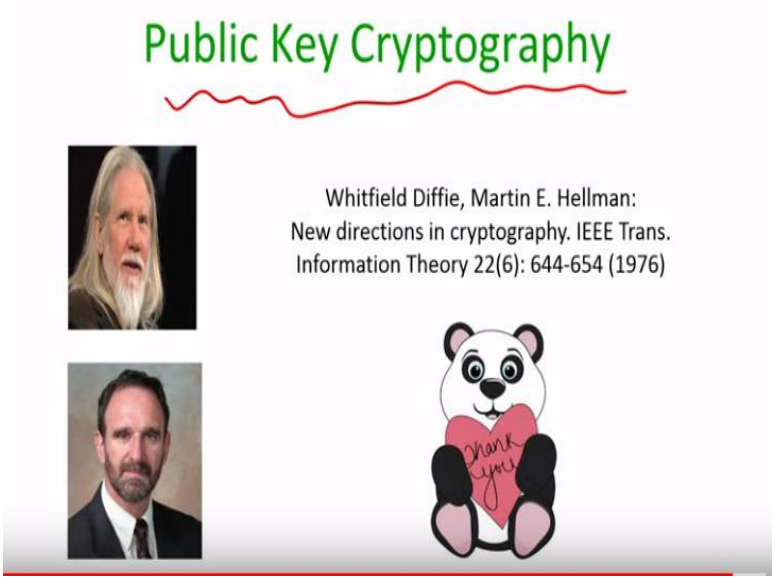
Whenever server would like to communicate something to the client, it runs an authenticated encryption scheme using the pair of keys ks and ks prime and we can use any authenticated encryption scheme here, say the one obtained by using the encrypt, then authenticate mechanism, and in response whenever client would like to communicate something to the server, it uses the other pair of key namely the key k sub c, and key k sub c prime.

You can recall that in the generic approach that we had seen to construct the authenticated encryption scheme, we stress that there should be an independent key for instantiating the CPS secure encryption scheme that is there inside the authenticated encryption scheme and there should be an independent key to instantiate the MAC component, which is there in the authenticated encryption scheme and that is why there is a pair of key, which is used in the authenticated encryption scheme.

We have a dedicated pair of key whenever server wants to communicate with the client and we have a dedicated key whenever the client wants to communicate to the server and now

you can see that for doing the actual communication between the client and the server, we are using completely symmetric key encryption process.

(Refer Slide Time: 27:45)



So to summarize, we should definitely thank these two genius, people with Diffie and Hellman who thought about to solve the key agreement problem whereas the sender and receiver who are connected by an in-secured channel to public communication and still can end up agreeing upon a common random private key known only to the sender and the receiver.

It is only because of their vision and due to the pioneer work, the whole area of public key cryptography emerged and that is how we could think of doing secured communication with any unknown person with whom we have never met, with whom we have never shared any kind of secret information and we can conveniently do secured communication with those unknown people.

So, we should definitely thank these two people because of the impact of work that what they have done, they are awarded during award very recently. That is all for this lecture. So just to summarize in this lecture, we have seen, we have discussed on a very high level that we have Fiat-Shamir heuristic and how we can apply it to convert any three-round identification scheme into a signature scheme. We applied it concretely for Schnorr identification scheme to obtain Schnorr signature scheme and we discussed a very high level overview of the SSL protocol and a TLS protocol.

So, that more or less ends our discussion on the problem of secured communication. We now how known how to do secured communication between two entities over a public channel where the sender and receiver may not have any pre-shared information. For the remaining lectures, we will see that how we can use cryptographic primitives to design interactive protocols where the goal is not just to solve the problem of secured communication, but rather to do something else. Thank you.